ADAPTING MODSAF APPLICATIONS TO COMPLY WITH THE HLA

Joshua E. Smith
Sagacitech, LLC
PO Box 337
Barre, MA 01005
jesmith@sagacitech.com
http://www.sagacitech.com

KEYWORDS

HLA RTI, Presentation Layer, ModSAF

ABSTRACT

The similarity between the HLA RTI and the existing (pre-HLA) DIS simulation paradigms makes porting applications fairly straightforward. This paper discusses one such port -- the ModSAF infrastructure. The described work forms the basis of the simulations underlying the STOW ACTD in 11/97. The paper will describe the manner in which the ModSAF software was modified to use the HLA, and will provide recommendations for changes to the RTI interface and/or implementation.

1. OVERVIEW

For the past several months, an effort has been ongoing to embed the HLA Run Time Infrastructure within the ModSAF software architecture. This work is in support of the DARPA/DMSO Synthetic Theater of War (STOW) program, and was sponsored by George Lukes under the Dynamic Virtual Worlds contract though Lockheed Martin.

The ultimate objective of this work is to create a version of the ModSAF architecture which can support STOW-scale exercises using the HLA RTI as the communications infrastructure. Along that path, we hoped to provide insights into the quality and nature of the RTI prototype implementation, its formal interface specification, and the impact of the HLA formalisms on simulation applications (memory usage, compute time, etc.).

After a few false starts down other paths, it is now clear that the ModSAF architecture forms the lowest risk basis for STOW simulation applications. This is because the vast bulk of STOW will be made up of synthetic forces and synthetic environment simulations which are based on the ModSAF architecture. Further, the only 3D viewport involved in STOW will be a stealth application which is also ModSAF-based. Thus, by getting ModSAF to support the HLA RTI, the vast majority of STOW is brought into compliance.

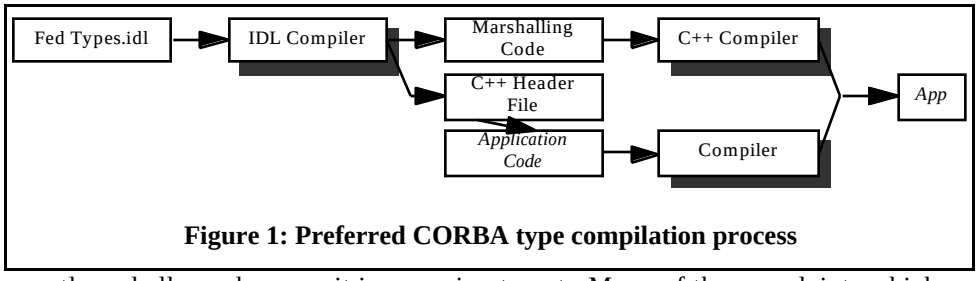

Figure 1: Preferred CORBA type compilation process

This paper will discuss the approach taken, technical details, and a hint at lessons learned. Criticizing the RTI is currently a challenge because it is a moving target. Many of the complaints which are discussed in this paper refer to the early-summer 1996 0.32E version of the RTI prototype implementation (which doesn't really correspond exactly to any particular version of the RTI I/F specification). Largely as a result of this work, the RTI development plans have been dramatically changed. Where possible, relevant RTI revisions will be noted.

2. TECHNICAL APPROACH

This section will discuss, at a high level, the project's technical approach. Following sections will highlight particular aspects of that approach.

2.1 General Goals

In part, the HLA RTI is a reengineering of a successful DARPA STOW program called RITN (Real-time Information Transfer and Networking). The RITN project demonstrated a set of "application control techniques" which included a consistency protocol which uses drastically less bandwidth than traditional DIS "heartbeats" and the use of wide area IP multicasting. The RITN program demonstrated the application of these technologies in late 1995, proving that they could significantly improve scalability.

Thus, one of the more pressing goals of the ModSAF/RTI integration is to demonstrate that nothing was lost in the reengineering process. Ideally, the new system should perform at least as well as RITN.

Along these same lines, we need to recognize that the DIS systems which the HLA intends to replace are themselves quite capable. In particular, time management, exercise management, and related functions must be exercised. The only major area which we elected to avoid in the first round was "Ownership Management." The HLA formalism of ownership management, in which individual attributes are handed off, rather than entire entities, is quite foreign to DIS. This makes the integration of these concepts with an existing DIS simulation quite a challenge. Combined with the realization that ownership management was not getting much attention from the RTI developers, we have elected to skip ownership management activities for STOW.

2.2 Insulating the Application

Because of the immaturity of the RTI software, and the continuing STOW need for development and testing, we took it as a precondition that continued interoperability with SIMNET and DIS must be maintained throughout the RTI integration effort. This took shape as follows:

- RTI dependencies have been limited to a handful of architectural libraries.

- All RTI dependencies are wrapped in compiler directives which allow them to be completely skipped in non-RTI environments.
- When RTI dependencies are compiled in, they must further be enabled at run-time via a command-line argument. This allows a single application to be developed and tested in DIS, then switched to RTI operation when necessary.

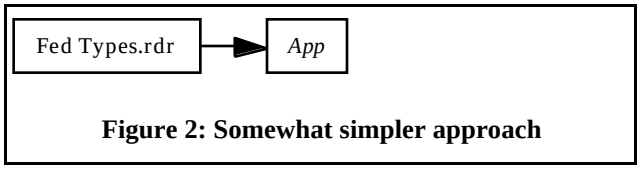

Figure 2: Somewhat simpler approach

The original version of the RTI used in this effort used the CORBA IDL specification language for its interface. Ironically, due to the immaturity of the technology, IDL turns out to be a really bad way of defining interfaces. Application developers need to choose an ORB vendor, which ties them to a particular set of supported platforms and target languages. Thus, portability and language independence are lost. This is in contrast to the traditional (portable, language independent) mechanism of creating an ANSI C API with Ada bindings. In addition, CORBA has arcane calling semantics which require excessive use of dynamically allocated memory, limited pass-by-reference arguments, no function pointers, and a complex way of returning status codes. What would generally be a one-line call to an ANSI C API will easily explode into a complex 20-line invocation to IDL-derived C++ interfaces. To avoid letting this complexity invade the ModSAF application, we wrapped the IDL interface in a simple ANSI C API. This API forms a strict boundary between the application and the RTI.

Another implication of the CORBA IDL approach is the need to perform strong compile-time definition of types. This leads to a process something like that of Figure 1.

In contrast, ModSAF uses a presentation layer service called PDU API which allows types to be bound at run time via a combination of function calls and data files. The advantage of run time binding is clear for ModSAF, where a single application might need to run with SIMNET protocol one day, and DIS 2.0.3 the next. Losing that capability with RTI integration would be unacceptable. Thus, we work around the strong typing requirements by telling the RTI that all complex types are merely "bags of bytes" which it should send unaltered. Since the RTI couldn't care less what the payloads of its messages mean, this does nothing to harm interoperability, and allows the use of the much simpler process shown in Figure 2.

2.3 The Illusion of Compliance

Although the above treatment of types is not in keeping with the CORBA philosophy, one would be hard pressed to say that it violates any compliance tenets of the HLA. However, some of the other simplifications which have been made in the near term are much easier to take issue with.

2.3.1 PO Protocol

ModSAF uses an abstraction called the "Persistent Object" database as the basis of much of its architecture. Synthetic organizations, their missions, and much of the simulation state is represented by objects in this database. The database is implemented using a protocol which is sort of a hybrid between the DIS heartbeat mechanism and the RITN Consistency Protocol supported by the RTI. Although we have every intention of migrating the PO database abstraction to use the native RTI communications services instead of this protocol, it will take time and money to get there. In the mean time, we continue to use the PO protocol as a back-door mechanism.

Strangely, this *is not* seen as a particularly non-compliant technique, since the number of ModSAF simulations which communicate using PO tends to be rather small (typically, a "farm" of 8 or fewer simulations will be pooled with a set of user interfaces on a single PO database). Thus, the collection of simulations using a single PO database are a single "simulation" by the definitions of the HLA, and all is right with the world.

2.3.2 SIMAN Protocol

Some of the simulation management (SIMAN) protocol messages are completely usurped by RTI functions (for example, "Pause"). However, the generic get/set attribute mechanisms provided by SIMAN are being used more and more by the DIS community as a flexible simulation protocol abstraction. ModSAF applications currently use SIMAN for stealth control, simulation hand-off coordination, and advanced resupply operations. In effect, these are new "Object Interaction Protocols" which are being implemented via attribute get/set operations.

To date, we have not worked out any sort of approach to migrating this functionality into an RTI implementation. Currently, it goes out the back-door as DIS messages, analogous to the PO traffic. Again, this kind of back-channel communication is both illegal under the HLA, and acceptable with some clever labeling of what is a "simulation." However, we have every intention of migrating this to strict compliance eventually.

2.3.3 Time Management

Time management in the HLA doesn't work for real-time simulation. Rather than supporting applications which try to be agile about time management (ask permission before advancing each frame), the developers have instead elected to just allow time management to be turned on or off. Fair enough – for ModSAF, and any other real-time simulation, we have to turn it off.

2.3.4 Ownership Management

As mentioned earlier, we are making no effort to use ownership management for STOW. We will have to confront this to a certain extent, since the hand-off operations (using SIMAN) and fault tolerant recovery (using PO) supported by ModSAF have ownership management implications. However, we will do only the minimum necessary to make the RTI function.

3. PDU'S TO FOM'S

One of the most significant changes between DIS and the HLA is merely one of terminology. What we used to call a "PDU," we now call a "Class." Little else is different, except that classes need to be characterized as either "Object" or "Interaction" classes. The choice makes an enormous difference in the RTI calls used to act on objects. Figure 3 shows the division made for this effort.

For this effort, the objective was to keep FOM development as simple and automated as possible. What that means is that each PDU was translated into either an Object Class or an Interaction Class. There is no inheritance. Note that for expediency, some classes were treated as Interaction classes, when they ultimately should become Object classes:

- Emissions are currently used by ModSAF for modeling Radar Warning Receivers. As such, they are most easily treated as interactions. However, since emitters have persistent state, they should really be treated as objects.
- Laser Designations are dealt with as objects within ModSAF, but not using the standard "vehicle table" approach, because they do not have IDs in DIS. This should be corrected.
- Environmental Change Notices (ECNs, for dynamic terrain) can be treated either as object or interactions. We believe that the most elegant solution would be to treat them as objects and exploit HLA ownership management, however this will likely not be attempted for STOW, because it has been deemed too high risk.

Object Class	**Interaction Class**
Entity State	Fire
Environmental	Detonation
Stealth State	Collision
Receiver	Service Request
Transmitter	Resupply Offer
Minefield	Resupply Received
Mines	Resupply Cancel
Aggregate State	Repair Complete
	Repair Response
Interaction pro tem	Signal
Emissions	
Laser Designator	
ECN	

Figure 3: DIS PDUs Acting Classy

3.1 Other Nuisances

The RTI interface defines a type for object IDs, which means that all occurrences of the traditional 48-bit site/application/entity ID within DIS PDUs must be changed to use this type instead. This was achieved by adding code to PDU API (ModSAF's presentation-layer abstraction) which performs this translation as the protocol binding files are read. This means that a single protocol binding file can be used either for DIS *or* to generate a FOM for the RTI. This is handy, since several ongoing efforts (synthetic environments, counter-mines, etc.) are developing new protocols (and PDUs). If these work in DIS, they will *automatically* work with the RTI as well. Note that the DIS convention of identifying radios by an Entity ID/Radio ID pair doesn't work with the RTI. The RTI assumes that all object IDs are drawn from a single, uniform space. Thus, PDU API automatically adds an RTI ID field to the transmitter PDU.

An additional data file informs PDU API of which fields in a PDU correspond to the sender/receiver arguments of the send-interaction call, and the object ID argument of the update-attributes call. This allows PDU API to eliminate these from the classes defined in the FOM, and to provide them separately to the application at run time so they can be passed properly.

One "gotcha" of the attribute-orientation of the RTI interface is that applications cannot assume that variable length arrays will be updated at the same time that the field which indicates their length will be updated. Thus, it is imperative that variable-length arrays be *clustered* into a structure with their length indicators. In the case of

optional fields, the field-presence indicator must be clustered with the field; for optional arrays, *both* other fields must be clustered. If a field indicates the length of more than one array (as the number-of-mines field in the proposed Mines PDU does), that field must be copied into all clusters. Like the translation of object IDs, this clustering is performed at run time by PDU API as the protocol bindings are being read, so that a single protocol can be used within DIS or with the RTI.

3.2 Interface Mechanics

The RTI interface is attribute-oriented. Rather than passing whole objects or interactions as structures, they are passed as arrays of name/value pairs. This is useful within the RTI, since different attributes of a single object might be passed using different transport mechanism, or even be updated by different computers! However, it is a lousy way for the application to see the data, since getting/setting the value of any given attribute would require a whole mess of search. Instead, we stick to a DIS PDU-like (with the modifications mentioned in the previous section) structure internally. The name/value pairs point into this structure, as shown in Figure 4. Note that this efficient use of dynamic memory (allocating one block, and pointing the value part of each pair into this structure – instead of allocating gobs of little blocks, one for each value) is supported through the ANSI C API we defined, but is currently not available in the IDL-derived interface. This is because CORBA does not allow passing pointers (unless every last bit of code on both sides of the interface is written in C++). Thus, the translation layer is forced to do a lot of malloc&copy operations to conform to the CORBA semantics.

Note: Efforts are ongoing at MIT Lincoln Lab to produce a non-IDL interface which can efficiently implement argument passing.

The generation of the name/value pairs is done by PDU API as it constructs network-format PDUs. Similarly, PDU API has functions which will extract the data in a name/value pair list and place it into a PDU-like format. This latter operation is generally done as an overwrite of a previous PDU, which allows asynchronous updates of random attributes to be merged into a single object structure.

Because PDU-like structures are used internally when interfacing to the RTI, the vast majority of PDU-speaking modules in ModSAF do not need to be modified at all to use the RTI. It just works. However, the five libraries which abstract "object class" PDUs did need to be modified slightly to handle new semantic rules which come with the RTI. In particular, the application modules can track which attributes are changing, to minimize network updates. Also, the RTI is responsible for triggering refresh messages ("heartbeats"), so the application modules must register for these "Please Provide Your Attributes" callbacks.

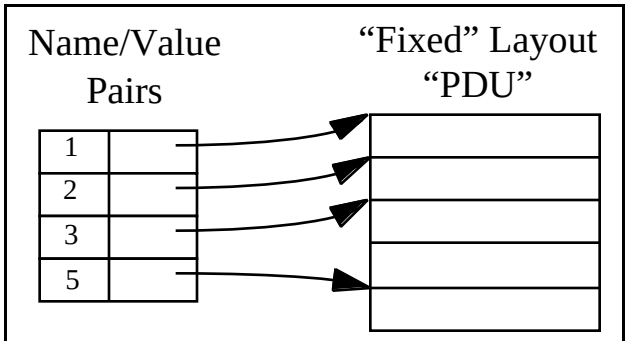

Figure 4: Pointing the RTI in the right direction

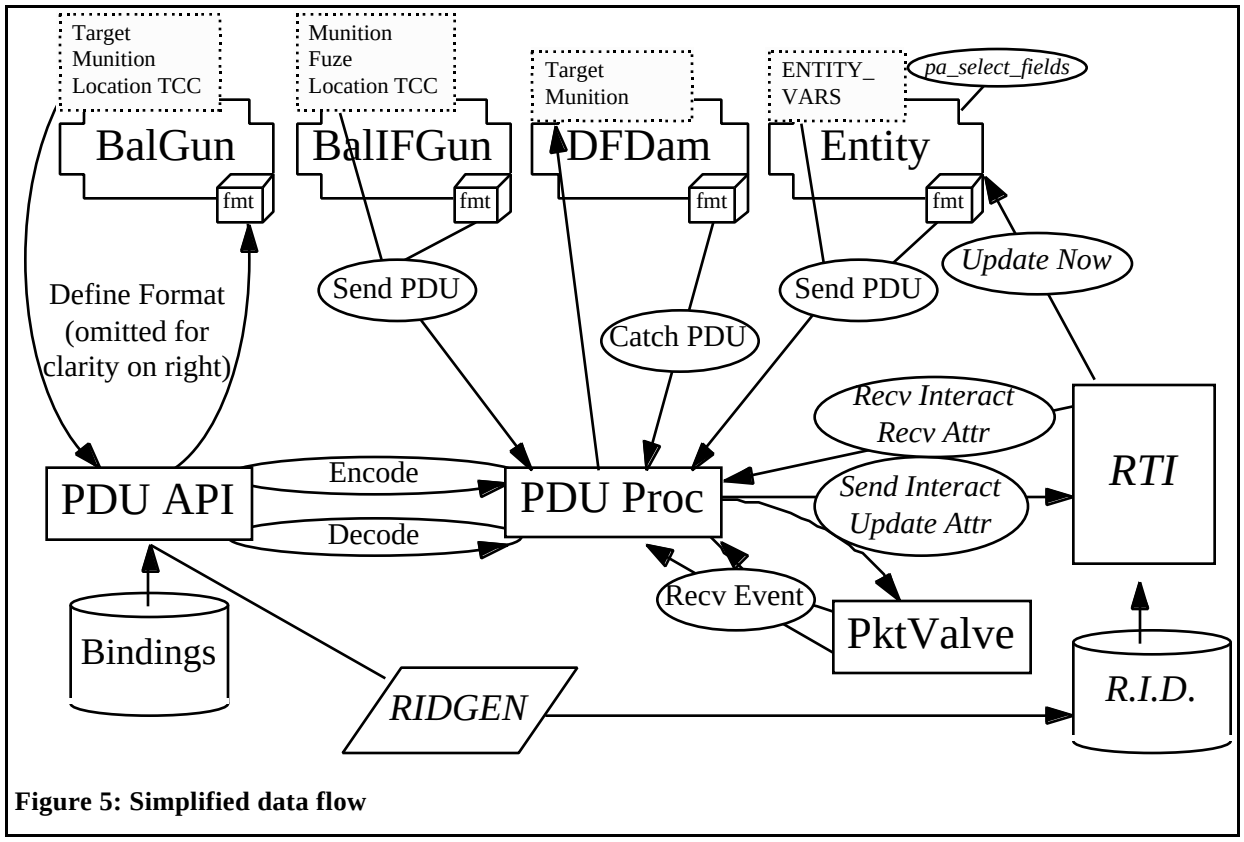

Figure 5: Simplified data flow

3.3 Integrated Data Flow

Figure 5 depicts the data flow for objects and interactions (the classes formerly known as PDUs) within ModSAF. The new pieces related to this RTI work are in italics. First, a summary of how things worked in DIS, then an explanation of what was added for the RTI:

· DIS

Starting at the left side of the diagram, note that BalGun (a ballistic gun model) knows some attributes of a detonation PDU in a particular presentation (coordinate system, enumeration document version, etc.). It passes this information to PDU API which looks at the Protocol Bindings which specify how that information is formatted in network PDUs. PDU API generates a small formatting program which can be interpreted at run time to quickly convert from the application's presentation to the network presentation.

When a module wants to send a PDU, it passes a pointer to its own data structure, and this "format" to a utility library called PDU Proc. This library asks PDU API to encode the information, and then sends it to the wire via PktValve.

PktValve may loop back this PDU to modules which have registered interest, in which case PDU Proc asks PDU API to decode it for each recipient and passes a decoded version on up. It also passes up the raw network data, which may be used later by the application module to further interpret the data.

Notably, most application modules do not allocate a new data structure in which to pass their data for encoding. Instead, they generally pass a pointer to a state structure of some kind which has the current values. This eliminates a lot of copying.

· RTI

When we use the RTI, in addition to PDU API marshaling the data into a PDU, it also generates a name/value pair list which can be passed to the RTI interface. The PDU is also sent to PktValve for possible loopback to the application, but it is tagged "Don't Send" so that it doesn't go to the wire.

On input, PDU API is used to "arrange" the data into a DIS-like PDU, which is then routed through PktValve. From there, it looks like any other incoming datagram, and the existing PDU Proc infrastructure for packet handling kicks in.

To deal with the complication of objects, which don't necessarily get every attribute, every update, PDU Proc keeps a buffer of the most-recent PDU for each object. Every update is folded onto this existing PDU, so that the application is always passed a complete PDU.

One more complication of objects is that the application now has the ability to issue only partial updates. This is done by calling a new PDU API function (pa_select_fields) which "rigs" the format to send only those attributes identified.

4. USING NEW RTI FEATURES

4.1 Use best transport choice per-attribute

The RTI makes available a number of new options for transporting object attributes. Almost all object attributes are best carried using a "state consistent" mechanism. This is similar to the traditional DIS heartbeat, except that the heartbeat is not a full refresh – it is a sequence number which rolls up the most recent updates of all attributes on a per-host/per-multicast address basis. (Scalable NAK mechanisms are used to resolve inconsistencies.) There are a few attributes (position, velocity, orientation), which benefit from a straight heartbeat while an entity is moving, but should fall back to a state consistent transport when the entity stops. Finally, best effort delivery is useful for attribute updates between the minimum heartbeat refreshes, as explained below.

4.2 Multiple-fidelity Attributes and Filter Spaces

Filter spaces (which may be renamed "Routing Spaces," just to keep people off balance) are an abstraction of multicast routing and forwarding. The idea is that a given federation will define a set of filter spaces, each of which corresponds to a segment of the communications traffic (a set of multicast addresses). A space can have an arbitrary number of dimensions, and subscriptions and publications are made to points or regions within these spaces. For this work, we started with three spaces:

Lo-Fi: This filter space is defined by geography, and carries almost all DIS-like information.

Hi-Fi: This filter space carries copies of the location, velocity, and orientation attributes of entities, which are transmitted best-effort whenever they exceed DR thresholds (between heartbeat updates, which are sent to the Lo-Fi space). To receive traditional DIS accuracy information, subscriptions to both the Lo-Fi and Hi-Fi spaces are required. Plan-view-displays, long-range sensors (e.g., JSTARS), and other lo-fi customers can just omit the Hi-Fi subscription.

Radios: This filter space is indexed by dimensions which isolate what a given radio might be able to receive (frequency, crypto, etc.). Since radios have such enormous potential ranges, geography is not among these dimensions. Transmitter and Signal information is sent via this filter space.

4.3 Matching RITN

In principle, the use of good transport choices, along with effect filter space definitions should allow the system to use the wide-area network as effectively as RITN did in the demo last year. Note that by dividing the attributes into differently-transported sets, we should achieve bit-rate reductions beyond those of RITN (whether we can avoid concomitant packet rate increases has yet to be proven).

5. RECOMMENDED RTI CHANGES

It was helpful throughout this endeavor to chant the mantra: "remember, it's just a prototype." At some point, however, we need to move beyond this fit-it-next-time philosophy. To facilitate this, a report has been prepared which details desired changes in the RTI interface and/or implementation. The full text of the report can be found at:

http://www.sagacitech.com/rtimods.doc

It appears that many of these suggestions have been (or soon will be) accepted by the HLA committees, although few are reflected in the current revision of the RTI interface spec. The comments which follow are in response to version 0.5 of the RTI I/F Spec, and version 0.32E of the RTI prototype implementation.

5.1 Grant time advance requests synchronously when time management stuff is "turned off"

5.1.1 Discussion of Issue

Although it is quite feasible for the RTI to hide the time management approach from the application, without any loss of performance, the current implementation does not do this. It should. One key element is granting time advance requests synchronously as they are made, for real time simulations.

As currently implemented, the RTI never invokes an application callback (that is, a method of the simulation ambassador) during the thread of an invocation of an RTI function (that is, a method of the RTI ambassador). This means a real time simulation has to do something like the following:

```
Begin Frame
Repeat:
    Get current real time clock value from system
    Determine real time passed since last frame
    Scale time passed by simulation rate to
    determine new Logical Time
    Request RTI Time Advance to new Logical Time
    Pass the thread to the RTI, allowing it to invoke
    callbacks
Until the Time Advance Grant is forthcoming
<execute one frame of simulation>
End Frame
```

This has two obvious problems: 1) The RTI is effectively given an extremely high priority, which it does not rightfully deserve; and, 2) The overhead of passing control to the RTI before every frame in a SAF simulation, in which there will typically be hundreds of frames per second, is extraordinarily high.

If the RTI were instead to grant the time advances as a synchronous action, then the thread could be passed to the RTI at a more appropriate rate.

The whole issue of the programming model is ignored in the current interface specification, so fixing this by modifying the I/F spec is not currently feasible.

5.1.2 Recommended Action

Issues of programming model, process model, and threads are far more serious than the semantic issues of function names and data types which are currently getting all the attention in the RTI I/F Spec. If the current implementation is a representative prototype, then one is forced to conclude that the "one RTI can fit all federates" hypothesis is wrong, since it grossly underperforms for real time simulations. However, it is not yet clear that the current prototype is at all representative of what a *well engineered* RTI could achieve. Thus, there appear to be two possible actions:

A) Add sections to the I/F Spec document explaining the various *different* programming, process, and thread models which are to be used for different sorts of federations. Clarify that for a real time federation, time advancement must occur synchronously with the requests for advancement. Clarify that the current discussion of time advancement as a mechanism to control message delivery *does not apply* to federates trying to achieve real time frame-based simulation.

B) Or, find a unified programming model which suits both real time simulation and discrete, event stepped simulations. Modify the RTI prototype implementation to use this programming model. One aspect of this new model would be that time advancement is synchronous with time advance requests.

Once the programming model has been specified, each functional interface will need clear specification of the following:

- *Other interface methods which may be invoked by this method*

  Specification of this is required, in order that the simulation developer can ensure that simulation-provided methods are reentrant in the defined cases.

- *Maximum time-to-complete which should be anticipated by the simulation*

  This information is required by fixed-frame, hard real time simulations, and is helpful in the design of variable-frame, soft real time simulations. If the time is indeterminate, or influenced by the execution time of simulation-provided methods, then this would be a description, rather than a hard cycle or microsecond count.

5.2 Don't "discover" objects until all subscribed attributes are known

5.2.1 Discussion of Issue

The RTI typically "discovers" an object after hearing only one attribute. This leaves the application responsible for tracking how many attributes have been learned, so that it can decide when it knows enough about an object to treat it as a part of the simulation. It would be far simpler if the application was not told of "discovery" until *all* the subscribed-to attributes are known by the RTI. (Another approach is to require that the application provide a minimum set of attributes for discovery, which would also be an adequate solution.)

5.2.2 Recommended Action

Section 4.3 currently has the statement:

> **The *Instantiate Discovered Object* service supplies an object ID, its class, and either a value for all attributes or those currently available as specified by the federation execution parameters.**

This sentence is ambiguous, at best. It is not at all clear what the phrase "as specified by the federation execution parameters" is referring to. It can't be "those currently available," since availability varies at run time. Thus, we must assume that the specification is of the "either...or." That is, the RID is supposed to indicate whether all attributes must be delivered on discovery, or whether the available set will do. By this reading, the current prototype RTI implementation is **non-compliant** with the I/F Spec.

Putting that aside, the idea that discovery of currently-available parameters would ever be useful to a simulation seems preposterous. What could a simulation possibly do with an object, for which only a few parameters are known?

We recommend that the above sentence be modified to read:

> **The *Instantiate Discovered Object* service supplies an object ID, its class, and ~~either a~~ values for all subscribed attributes ~~or those currently available as specified by the federation execution parameters.~~**

and that the prototype RTI implementation be brought into compliance with this specification.

5.3 Define API in ANSI C with Ada Bindings - CORBA/IDL is too inefficient and clumsy

5.3.1 Discussion of Issue

The use of CORBA IDL is a very poor choice for real-time, high-performance simulation. We cannot afford the computational, nor the semantic, complexity of CORBA. Nor can we afford the usurious license fees being charged for the CORBA compilers.

There is a very old, and very effective way to define functional interfaces like the one of the RTI. You simply use ANSI C, which, unlike CORBA/IDL, is a widely available international standard. Ada programs (even Ada '83!) can easily bind to ANSI C declarations using a number of different techniques (pragma's, bindings, etc.). Similarly, other computer languages always provide a mechanism to call out to ANSI C. All major reusable software modules defined in the last ten years (X windows, POSIX, Open GL, etc.) use ANSI C as their API definition language. The choice of CORBA was apparently driven by bandwagon-think, not sound engineering.

5.3.2 Recommended Action

Rewrite the interface in ANSI C.

5.4 Relook the types selected.

5.4.1 Discussion of Issue

Many of the data types currently used within the RTI are short-sighted. For example, a simulation ID is currently only 16 bits. To support the use of the RTI in a large-scale, heterogeneous environment (possible including scalable multi-processors with thousands of nodes), this simply isn't enough bits. Similarly, the use of floating point to represent time (time is more accurate earlier in the century), is an odd choice, and does not conform to existing standards like POSIX (which uses a structure with seconds and microseconds, expressed as 32 bit integers).

5.4.2 Recommended Action

Change Object ID Count type to 32 bits.

Change Simulation ID type to 32 bits.

Change Time representation to POSIX standard.

Review other types to determine if they really make sense. In particular, note whether they should be signed or unsigned quantities.

5.5 Either let the application format the data, or build a much smarter data cruncher in the RTI

5.5.1 Discussion of Issue

Currently, the assumption is that the RTI should be responsible for marshaling the data for placement on the network. However, the RTI is not given enough information in the FOM (say, required bits of resolution) to do this effectively. Furthermore, it will often be the case that the application can pack the bits much more efficiently than the RTI. We are currently doing this with ModSAF, to avoid IDL compilation steps and avert known bugs in the RTI implementation, but many have frowned on this as "subverting" the HLA. We believe that this either should be formally recognized as a compliant use of the HLA RTI, or that the RTI interface should be redesigned to allow it to efficiently pass the data on the network - preferably, both.

5.5.2 Recommended Action

Clarify data marshaling expectations and responsibilities in the RTI I/F Spec.

5.6 The RTI should provide Simulation IDs, not the federate

5.6.1 Discussion of Issue

For some reason, the RTI interface requires that the federates provide their own simulation IDs (a.k.a. federate ID). Since these IDs are invariably derived from network addresses, it really should be province of the RTI to select these IDs. Also, it would be more consistent with the treatment of Object IDs to have the RTI generate simulation IDs.

5.6.2 Recommended Action

In section 2.3 move “Federate ID” from the **Supplied Parameters** to the **Returned Parameters**.

5.7 Restore should require "Simulation Paused" as a precondition

5.7.1 Discussion of Issue

The current interface spec does not require that the simulation be paused prior to restoring state. It should. Our experience with ModSAF is that trying to restore state with the clock running is next to impossible.

5.7.2 Recommended Action

In section 2.16 add “Federation is paused” to the list of **Pre-conditions**.

5.8 Parameters of interaction classes should be passed as name/value pairs, to allow omission of irrelevant or defaulted values

5.8.1 Discussion of Issue

There is an inconsistency in the current RTI implementation, in that attribute updates are passed as name/value pair arrays, but interactions are passed as value arrays only (no names). This would require twice as much code in some parts of the application, to deal with the two data structures. Also, in principle, interactions should be able to omit parameters which do not apply or can be assumed to have a default value. This would only be possible if they are passed as name/value pairs. The ANSI C API (suggested above) presents a consistent interface for both object and interaction classes, which thus requires that the translation layer regenerate the name on the way in.

5.8.2 Recommended Action

Change the RTI implementation to present a consistent interface.

5.9 Get rid of "with user time" variations of many functions

5.9.1 Discussion of Issue

Many methods have a separate "with user time" variation for no apparent reason (since all the impacted methods carry a logical time stamp independent of this user time). If a federation needs to attach a special time to its classes, it should put it into the FOM. There is no justification for treating this attribute any differently. Also, by making a separate version of these functions, interoperability is hampered, since applications which use the "with user time" variation will not work with applications which use the simpler version - and this distinction will not be captured in the FOM!

5.9.2 Recommended Action

Eliminate “Optional User Supplied Time Stamp” from parameters of the following functions. Also eliminate with_user_time variations in the API:

4.3 Instantiate Discovered Object †
4.4 Delete Object
4.5 Remove Object †
4.6 Update Attribute Values
4.7 Reflect Attribute Values †
4.9 Send Interaction
4.10 Receive Interaction †

5.10 Allow network topology-based subscription for distributed logging

5.10.1 Discussion of Issue

The current RTI interface spec is actually very, very good for writing data loggers/players. However, it is lacking a mechanism whereby distributed logging can be done without adding to the LAN traffic rates. The idea is that a logger could subscribe to everything which comes from the Local LAN, but rely on other loggers to record information from other LANs. Although this can be done by fiddling with the filter space definitions, it would be far better for the RTI to support this kind of subscription directly.

5.10.2 Recommended Action

There is currently an active debate on the best way to implement this functionality without having catastrophic impacts on the effectiveness of multicast. This is clearly one area where the implementation should drive the interface, so we advocate choosing whatever interface is most natural to the selected implementation.

5.11 Provide a way to learn execution state

5.11.1 Discussion of Issue

When a federate joins a federation, there is currently no way for it to find out some crucial details, like whether the execution is currently running. Either query functions (analogous to the available “get federation time”) should be provided, or the RTI interface should guarantee that the application will be called-back immediately to inform it of the current state (for example, by invoking the "Resume Now" callback if the execution is not paused).

5.11.2 Recommended Action

Provide one of the above solutions — either a query for current run state, or an guaranteed invocation of the Pause Now or Resume Now methods at startup time.

5.12 Eliminate strange distinction between simulation rate and running/paused.

5.12.1 Discussion of Issue

The RTI interface makes a distinction between the rate of the federation (used when running at scaled-real-time), and whether the federation is running or paused. There is no sense in this distinction. A paused simulation is simply one which is running at zero times real time (or the discrete-event equivalent - a stalled clock).

5.12.2 Recommended Action

Eliminate Pause, Resume, Pause Now, and Resume Now interfaces.

5.13 Provide a shorthand for saying "all attributes" in all attribute-oriented functions.

5.13.1 Discussion of Issue

Many RTI interface functions take attribute lists as their arguments. In many cases, the application merely wants to say "all attributes," but to do so requires the construction of a comprehensive list. This should be optimized, such that the application can just pass a special token which implies all attributes (for example, have an empty attribute list imply all attributes).

5.13.2 Recommended Action

In the case of “Subscribe Attribute Values” this option is in the I/F Spec, but the current RTI implementation is **non-compliant**, in that it does not present it in the API. In other cases, this will need to be added to the I/F Spec.

6. CONCLUSIONS

The migration of DIS applications to use the HLA RTI will be a long and painful process. It is likely that many applications will not have sufficient presentation layer abstractions to make it feasible to use the RTI except by way of translators (the PDU API abstraction in ModSAF was added just last summer). Fortunately, the DARPA STOW program is out in front, forcing "big program reality" into the RTI development plans, and finding the pitfalls and solutions before everyone else climbs on. DMSO has proven quite agile at adjusting their plans to better accommodate STOW's requirements. This bodes well for the DIS community, but continued vigilance is still the best policy.